ソフトウェア工学　136－18
（２００２．３．８）

組込みソフトウェア開発における XML を用いた再利用支援方式

岡　本　鉄　兵　　小　泉　寿　男

東京電機大学大学院理工学研究科

本論文では，XML を用いてソフトウェア部品の再利用を支援する組込みソフトウェアの開発方式を提案する．本方式では，設計対象を構成するソフトウェア部品の組合せを記述するために XML 規格に準拠した仕様記述言語 STML を開発した．近年，組込みシステムの応用分野の多様化と，それを支える基盤技術の進歩により，組込みシステムのソフトウェア（組込みソフトウェア）の需要は増大傾向にある．組込みソフトウェアは，製品に実装されてターゲットを制御するため，リアルタイム応答や並列処理，メモリ容量など，様々な制約がある．また，製品の高機能化も伴って，組込みソフトウェアは，複雑かつ巨大なものとなり，開発生産性の低下を招く要因となっている．本提案方式を用いることによって，ソフトウェア部品の再利用を支援する組込みソフトウェア開発を実現し，開発生産性の向上を目指す．

An XML-based Development Method of Re-use for Embedded Software

TEPPEI OKAMOTO　and　HISAO KOIZUMI

Graduate School of Science and Engineering, Tokyo Denki University

In this paper, we propose a development method of embedded software to support reusing software components by using XML. We have developed STML (State Transition Markup Language) which is a specification description language based on XML (eXtensible Markup Language) and markups the structure of many components on design object. Recent year, software for embedded systems is in great demand because of those diversification and advancement of technology. The software is mounted on embedded products and controls the systems, then it has many restrictions in real-time reply, parallel processing, storage capacity and so on. And embedded systems are requested to be high performed, and so the embedded software is more complicated and larger in size. It is a factor of low productivity. Using the method, we aim to realize the embedded software development fit for reusing components, and aim to improve the productivity.

1. はじめに

組込みソフトウェア（Embedded Software）は，家電製品や自動車，情報通信機器，FA など，各種産業機器に実装される制御用ソフトウェアである．近年，半導体技術の発展に伴い，安価で高機能なマイクロプロセッサが登場し，組込みソフトウェアの応用分野は広がった．また，要求されるソフトウェアも複雑かつ大規模なものとなった結果，組込みソフトウェアの開発効率と品質の低下を招くことになり，その改善が課題となっている．

効率的なソフトウェア開発を目指すとき，ソフトウェア部品の再利用がその突破口となる．オブジェクト指向に基づくソフトウェアの部品化技術としては，Gamma らのデザインパターン [1] をはじめ，アナリシスパターン [2]，プラグ&プレイ型のソフトウェア部品であるコンポーネントウェア [3]などが代表的である．

本論文では，XML を用いてソフトウェア部品の再利用を支援する組込みソフトウェア開発方式を提案し，プロトタイピングにより検証する．本方式は，再利用の対象となる分野別中枢フレームワークであるスケルトンと，詳細な機能別に分類した機能部品の組合せを設計情報として状態遷移表を用いて表現

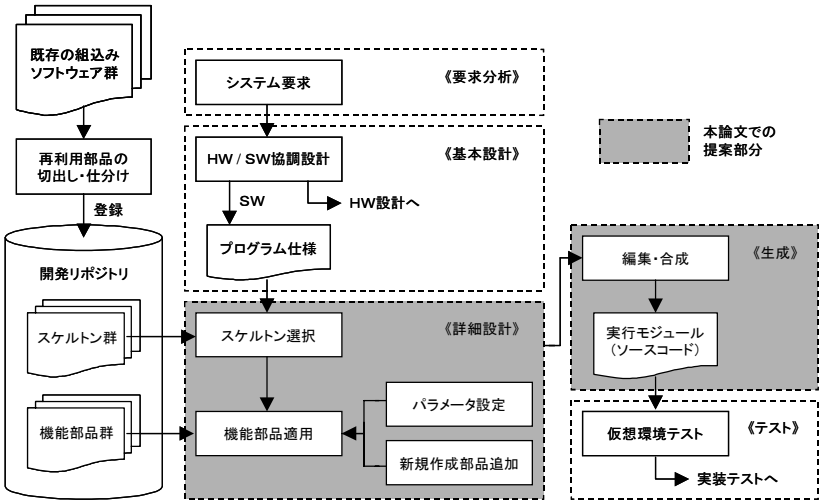

図 1　組込みソフトウェアの再利用型開発方式

する．これによって得られた設計情報は，XML 記述を用いて編集・管理される．設計情報を XML 化することによって，複数の状態遷移表の合成および部分的な切出しなどの編集作業が一元化されたデータとして取扱うことが可能となり，ソフトウェア部品の再利用を支援する．

2. 機能別ソフトウェア部品の再利用によるソフトウェア開発

著者らは，これまで機能別ソフトウェア部品の再利用を中核とした組込みソフトウェア開発方式を提案してきた[4)5)]．**図 1** に本方式の全体像を示す．本方式は，既存の組込みソフトウェアを分析し，再利用可能な分野別中枢フレームワーク（スケルトン）と機能部品を各々切出して，開発リポジトリに予め蓄積させておくことを前提として実現する．

新たに組込みソフトウェアを設計するときには，はじめに設計対象ソフトウェアの骨組みとなるスケルトンを開発リポジトリから選択し，次に選択したスケルトンに適用可能な機能部品を開発リポジトリから選択して，スケルトンに填め込んでゆく（**図 2**）．適用した機能部品に必要なパラメータを付加し，新規作成部品を追加した上で，それらを編集・合成して目的とする組込みソフトウェアの実行モジュール（ソースコード）を生成する．ここでのソフトウェア部品は，あらゆる分野で再利用可能な汎用性の高い部品というよりは，むしろ応用分野毎に分類し特化したソフトウェア部品である．このように分野毎に特化した部品を用いることは，部品の組合せによって必要とする機能を詳細に表現することが可能と

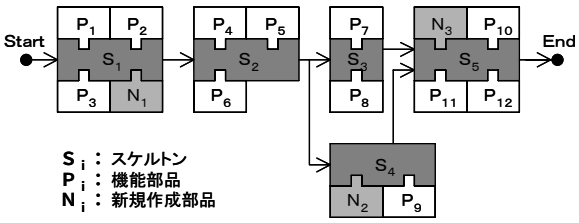

図 2　部品合成方式概念図

なり，新規に作成しなくてはならない部品を最小限に抑えてソフトウェア部品の再利用率を高めることを目指す．

生成した実行モジュールは，仮想環境上でテストした後に実装テストを行なう．このようにして，分析・設計・生成・テストという各開発段階を一貫させたトップダウン型開発方式を実現し，組込みソフトウェアの開発生産効率の改善を目指している．本論文では，詳細設計から生成に至る段階までを支援する開発方式を提案する．

3. STML を用いた再利用型開発方式

3.1　仕様記述言語 STML

再利用の対象となるスケルトンと機能部品を選択し，これらの部品同士の組合せ方を記述するために仕様記述言語 STML（State Transition Markup Language）を開発した．STML は，著者らが開発した組込みソフトウェアを対象とする動的仕様記述言語 EML（Embedded Markup Language）[5)] を仕様段階から見直し，拡張したものである．STML の開発にあたっては，次の点を考慮した．

① 実装（現実）と設計（論理）とを分離させた記述が可能であること

② 仕様変更に合わせて部分的な追加・削除が容易であること
③ 設計の抽象度に合わせて階層構造表現が可能であること
④ 記述内容をデータベース化し，再利用することが容易であること
⑤ 記述内容のコンピュータによる処理系の開発が容易であること

これらの条件からSTMLは，ネットワークに対応した標準データフォーマットとして現在注目されているXML (eXtensible Markup Language) 規格を用いて開発した．こうしたことによってSTMLは，アプリケーション間の汎用性が高く，ネットワーク化された分散開発環境にも対応した仕様記述言語を実現し得ると考える．

3.2 STML記述の基本仕様

STMLは，設計対象ソフトウェアの動的な振舞いを仕様化して表現するための記述言語であり，状態遷移表 [6]と同等の記述力を持たせることを開発指針とした．これによってSTMLにおける仕様記述は大きく分けると，状態・事象・アクション・遷移の4つの基本情報から構成され，以下の規則に従って記述される．

① **状態：**設計対象ソフトウェアは，有限個の状態が存在し，状態の抽象度によって階層化される．STMLでは，これを**<state>**タグを用いて表現する．
② **事象：**ある状態 ***X*** の下で起こり得る事象は，事象毎に**<event>**タグを用いて表現し，状態 ***X*** を表す**<state>**タグで囲む．
③ **アクション：**ある事象 ***a*** が発生したとき，実行されるアクションは，事象 ***a*** を表す**<event>**タグの **action** 属性で指定する．
④ **遷移：**ある事象 ***a*** が発生し，事象 ***a*** に関連付けられたアクションが実行されると状態は遷移する．遷移先となる状態は，事象 ***a*** を表す**<event>**タグの **trans** 属性で指定する．

3.3 STMLの記述例

動的な振舞い記述の例を**図3**, **図4**, **図5** に示す．

図3 は，状態遷移図（State Chart）の記述例であり，状態 **State1** の下位状態として状態 **S1**, 状態 **S2**, 状態 **S3** が階層化されており，状態 **State1** がアクテ

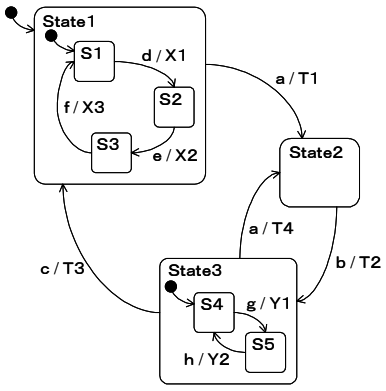

図3 状態遷移図の記述例

Table1

S \ E	a	b	c
State1	T1 ⇒State2		
State2		T2 ⇒State3	
State3	T4 ⇒State2		T3 ⇒State1

Table2

S \ E	d	e	f
S1	X1 ⇒S2		
S2		X2 ⇒S3	
S3			X3 ⇒S1

親状態：State1

Table3

S \ E	g	h
S4	Y1 ⇒S5	
S5		Y2 ⇒S4

親状態：State3

図4 状態遷移表の記述例

```
<state name="State1" activity="Activity−State1">                    (a)
    <event name="a" action="T1" trans="State2"/>
    <event name="b" action="none" trans="none"/>
    <event name="c" action="none" trans="none"/>

    <state name="S1" activity="Activity−S1">                        (b)
        <event name="d" action="X1" trans="S2"/>
        <event name="e" action="none" trans="none"/>
        <event name="f" action="none" trans="none"/>
    </state>

    <state name="S2" activity="Activity−S2">
        <event name="d" action="none" trans="none"/>
        <event name="e" action="X2" trans="S3"/>
        <event name="f" action="none" trans="none"/>
    </state>

    <state name="S3" activity="Activity−S3">
        <event name="d" action="none" trans="none"/>
        <event name="e" action="none" trans="none"/>
        <event name="f" action="X3" trans="S1"/>
    </state>
</state>

<state name="State2" activity="Activity−State2">
    <event name="a" action="none" trans="none"/>
    <event name="b" action="T2" trans="State3"/>
    <event name="c" action="none" trans="none"/>
</state>

<state name="State3" activity="Activity−State3">
    <event name="a" action="T4" trans="State2"/>
    <event name="b" action="none" trans="none"/>
    <event name="c" action="T3" trans="State1"/>

    <state name="S4" activity="Activity−S4">                        (c)
        <event name="g" action="Y1" trans="S5"/>
        <event name="h" action="none" trans="none"/>
    </state>

    <state name="S5" activity="Activity−S5">
        <event name="g" action="none" trans="none"/>
        <event name="h" action="Y2" trans="S4"/>
    </state>
</state>
```

図5 STMLの記述例

ィブなときには，状態 **S1**，**S2**，**S3** のいずれかひとつもアクティブになる or 状態が存在することが表現されている．状態 **State3** についても同様である．

図 4 は，**図 3** の状態遷移図を状態遷移表に置換したものであり，状態の階層化は，同一の状態階層毎に状態遷移表を作成して階層化することで表現している．状態遷移表は，状態と起こり得る事象の全てのケースを検出できるので，上流設計において，異常ケースの検出漏れが原因で作り込まれる誤りの削減にも有効である．

図 5 は，**図 4** の状態遷移表を STML で記述したものである．STML 記述の基本構造は，前節の通りである．STML では，状態の階層化を**<state>**タグの入れ子構造で表現するため，状態 **State1** を表す**<state>**タグで囲まれた中に状態 **S1**，**S2**，**S3** を表す**<state>**タグが入れ子になって表現されている．このように STML は，状態遷移表との間に互換性がある．

3.4 STML を用いたソフトウェア部品の再利用

本提案方式では，STML を用いて**図 2** で示した機能別ソフトウェア部品の再利用を実現する．再利用の対象となるソフトウェア部品は，状態遷移表の構成をスケルトンとして扱い，状態遷移表のセルに当て嵌めるアクションと，ある状態の下で実行されるアクティビティを機能別部品として再利用する．

例えば，**図 4** で示した状態遷移表群では，状態遷移表 **Table1**，**Table2**，**Table3** の構成をそれぞれスケルトンとして扱う．また，状態遷移表 **Table1** において，状態 **State1** の下で事象 **a** が起きたときのセルには，アクション **T1** を実行し，状態 **State2** に遷移することが記述されている．このときのアクション **T1** および **State1** の下で実行されているアクティビティ **State1** を機能部品として扱う．このような方法を採用したことによって，ソフトウェア構成をテンプレートとして再利用することと，個別の機能に対するソフトウェア部品を再利用することが同時に実現可能となる．

次に STML で記述された状態遷移表について論じる．STML 記述において 1 つの状態遷移表は，同一状態階層に存在する**<state>**タグ群と，その**<state>**タグの子である**<event>**タグ群である．ただし，同一状態階層に複数の状態遷移表が存在する場合は，親の状態毎に分割して考える．

例えば，**図 5** で示した STML 記述の場合，同一階層に存在する状態群 **{State1, State2, State3}** と，その子である事象群 **{a, b, c}** からなる状態遷移表 **(a)** が最上位の状態遷移表となる．一階層下位の状態階層には，状態 **S1, S2, S3, S4, S5** が存在し，**State1** を親状態とした状態群 **{S1, S2, S3}** と，**State3** を親状態とした状態群 **{S4, S5}** の 2 つの状態群に分類できるため，これらを 2 つの状態遷移表に分割する．これに従うと，状態遷移表**(a)** の直下の状態遷移表は，状態群 **{S1, S2, S3}** および事象群 **{d, e, f}** からなる状態遷移表**(b)** と，状態群 **{S4, S5}** および事象群 **{g, h}** からなる状態遷移表**(c)** の 2 つに分割される．

STML を用いることによって，複数の状態遷移表を結合することと，結合された状態遷移表を部分的に切出して再結合することが一元的な設計データとして管理できるようになる．この機能は，新しいスケルトンの切出しや，スケルトンのカスタマイズを行なって再利用の高いスケルトンを生成することを支援する．また STML は，XML 仕様に準拠しているため，設計情報をネットワークでやり取りし，データベース化することにも利用可能である．

```
/* STATE ID */
enum sttyp {State1,State2,State3} state;

/* EVENT ID */
enum evtyp {a,b,c} event;

/* 状態遷移表の駆動 */
state = State1;                         /* 初期状態 */
while(1){                               /* 無限ループ */
        event = get_event();            /* 事象確保 */
        switch(state){
        case State1:
                switch(event){
                case a:
                        T1();           /* アクション T1() を実行 */
                        state = State2; /* 状態 State2 へ遷移 */
                        break;
                case b:
                        none();         /* 何もしないときの処理 */
                        break;
                case c:
                        none();
                        break;
                }
                break;
        case State2:
                switch(event){
                case a:
                        none();
                        break;
                case b:
                        T2();           /* アクション T2() を実行 */
                        state = State3; /* 状態 State3 へ遷移 */
                        break;
                case c:
                        none();
                        break;
                }
                break;
        case State3:
                switch(event){
                case a:
                        T4();           /* アクション T4() を実行 */
                        state = State2; /* 状態 State2 へ遷移 */
                        break;
                case b:
                        none();
                        break;
                case c:
                        T3();           /* アクション T3() を実行 */
                        state = State1; /* 状態 State1 へ遷移 */
                        break;
                }
                break;
        }
}
```

図 6 STML のコンパイル例

3.5 STML のコンパイル方法

図 6 に STML で記述された設計情報のコンパイル例を示す．この例は，**図 5** における最上位の状態遷移表**(a)**の C 言語による実装を二重の switch - case 文を用いることで実現したものであり，最も基本的なコンパイル例である．また，switch - case 文を if - else 文に置換しても同じように実現できる．

STML のコンパイルは，C 言語以外の言語でも可能である．例えば，Java や C++などのオブジェクト指向言語による実装を行なう場合，状態遷移表を構成するセル単位のオブジェクトとして実装する State Table パターン[7]を用いる方法などが考えられる．また，コンパイラを自作しなくとも，有限状態機械のコンパイラは，シェアウェア，フリーウェアを含めて様々なものが公開されているので，これらを用いて実装を行なうことも可能である[8]．

以上のように STML は，ソフトウェア設計におけるメタデータを記述し，STML で記述されたひとつの設計情報から必要とする実装言語別のコンパイラを用いることによって複数の言語にコンパイルすることができる．このようなコンパイル方法を採用することによって STML は，設計と実装を分離させた仕様記述を実現する．

4. STML を用いた開発プロセス

前章での議論を踏まえて，**図 7** に STML を用いた組込みソフトウェア開発プロセスを示す．

詳細設計フェーズでは，基本設計フェーズで確定した組込みソフトウェア仕様に基づいて，設計対象となるソフトウェアのフレームワークとして再利用可能なスケルトンを選択する．選択されたスケルトンは，未完成な状態の状態遷移表として得られる．このとき，リポジトリに蓄積されているスケルトンが再利用できない場合は，新たに状態遷移表を一から作成し，必要に応じて新規のスケルトンとしてリポジトリに登録する．尚，リポジトリ内のスケルトンは，STML データとして保存する．

次に，選択された状態遷移表を構成するセル毎に対して実現すべき機能および処理を再利用可能な機能部品リストから選択し，セルに填め込んで状態遷移表を完成させる．このとき，リストにある部品が適用できない個所については，新規作成部品として新たに設計する．こうして完成した状態遷移表は，STML 文書の形で次の開発段階である生成フェーズに引き渡される．

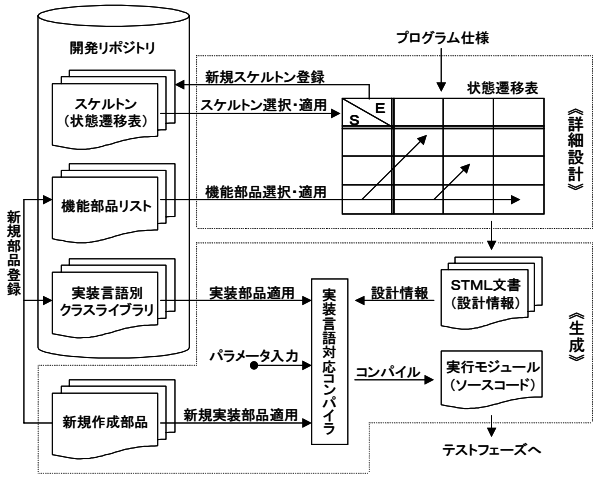

図 7 STML を用いた組込みソフトウェア開発方式

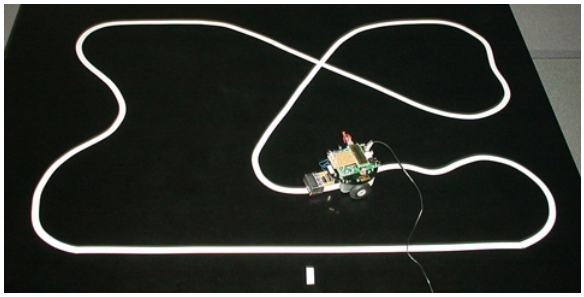

図 8 ライントレースシステムの全景

生成フェーズでは，詳細設計フェーズから引き渡された STML 文書に基づき，実装対象に対応した言語別クラスライブラリから必要な機能部品を適用する．また，新規作成部品が必要な場合は，この段階で新たに作成し，必要に応じて新規機能部品としてリポジトリに登録する．適用する機能部品と，新規作成部品およびこれらの部品の適用に必要なパラメータなどを設定した上で，コンパイラによってこれらを編集・合成して実行モジュールを生成する．生成された実行モジュールは，次の開発段階であるテストフェーズに引き渡され，仮想環境テストを経てから実機テストへと移される．

5. 実験と評価

5.1 ライントレースシステムの概要

本論文で提案した組込みソフトウェアの再利用型開発方式を用いて，マイクロマウスロボットによるライントレースシステム（**図 8**）を構築した例を示し，本方式の検証を行なう．

実験では，マイクロマウスロボット(Rug Warrior Pro) 用のライントレースシステムを構築した．このシステムは，マイクロマウスロボットが黒い床面に

表 1　準備した再利用を対象とした機能部品

関　数	動 作 内 容
void motor_drive(int operation)	operationに従ってマイクロマウスロボットのモータを駆動する.
int get_event()	ラインセンサから白線の検出状況を取得する.

表 2　ライントレースシステムの状態遷移表

S ＼ E		コースアウト	ライン中央(1)	ライン中央(2)	左に逸れる(小)	右に逸れる(小)	左に逸れる(大)	右に逸れる(大)	エラー
		0	1	2	3	4	5	6	7
停止	0	停止	前進	前進	前進	前進	右カーブ	左カーブ	エラー処理
		⇒State0	⇒State1	⇒State1	⇒State1	⇒State1	⇒State5	⇒State4	
前進	1	右回転	前進	前進	前進	前進	右カーブ	左カーブ	エラー処理
		⇒State3	⇒State1	⇒State1	⇒State1	⇒State1	⇒State5	⇒State4	
左回転	2	コースアウト処理	前進	前進	前進	前進	右カーブ	左カーブ	エラー処理
			⇒State1	⇒State1	⇒State1	⇒State1	⇒State5	⇒State4	
右回転	3	コースアウト処理	前進	前進	前進	前進	右カーブ	左カーブ	エラー処理
			⇒State1	⇒State1	⇒State1	⇒State1	⇒State5	⇒State4	
左カーブ	4	コースアウト処理	前進	前進	前進	前進	右カーブ	左カーブ	エラー処理
			⇒State1	⇒State1	⇒State1	⇒State1	⇒State5	⇒State4	
右カーブ	5	コースアウト処理	前進	前進	前進	前進	右カーブ	左カーブ	エラー処理
			⇒State1	⇒State1	⇒State1	⇒State1	⇒State5	⇒State4	

白線を引いたコース上で，白線をラインセンサで感知しながら，白線に沿ってコースを走行するものである．プログラムの実装環境は，Rug Warrior Pro に標準で用意されている Interactive C を用いた．

5.2　ライントレースシステムの構築実験と評価

実験を始める前に，再利用可能な機能部品を作成した（**表 1**）．作成した部品は，マイクロマウスロボットを停止・前進・後退・左回転・右回転・左カーブ・右カーブさせるための関数と，ラインセンサから白線の検出状況を取得するための関数である．

次に構築するライントレースシステムの主要動作部分となる状態遷移表を作成した（**表 2**）．この構築例では，ロボットの動作を状態，ラインセンサからの入力を事象として状態遷移表を作成し，STML で記述した．また，準備した機能部品を適用できなかった個所（コースアウト処理およびエラー処理）については，新たに機能部品を作成した（**表 3**）．

完成した STML 文書をコンパイルして実行モジュールを生成した．プログラムの主要部分となる状態遷移表の駆動については，Interactive C が switch - case 文に対応していないため，if - else 文を二重に用いた方法で実装させた．生成した実行モジュールは，マイクロマウスロボットにダウンロードした後，実際に実験用コースを走行させて動作を確認し，必要に応じて修正してソフトウェアを完成させた．

表 4 に実験結果をまとめる．生成された実行モジュールは，プログラムの主要部分となる状態遷移表の駆動部分をはじめ，主にソフトウェア部品を再利用して構成されている．プログラムの修正個所は，

表 3　新規に作成した機能部品

関　数	動　作　内　容
int course_out()	マイクロマウスロボットがコースアウトしたとき，直前の事象履歴を参照してコースに復帰するようにモータの駆動方法を決定する.
int input_error()	ラインセンサからの入力がエラーのとき，直前の事象履歴を参照して次のモータの駆動方法を決定し，エラーを回避する.

表 4　完成した実行モジュールのソフトウェア構成

関　数	分　類	動 作 内 容	利用形態
void motor_drive()	機能備品	**表1**参照.	再利用
int get_event()	機能備品	**表1**参照.	再利用
void STM()	スケルトン	状態遷移表を駆動する.	再利用
int course_out()	機能備品	**表3**参照.	新規作成
int input_error()	機能備品	**表3**参照.	新規作成
void main()	その他	プロセスを定義する.	新規作成

マイクロマウスロボットのモータの回転速度を決めるパラメータ部分を主に修正するのみで，動作した．

6. む す び

本論文では，設計情報を XML 記述によって管理し，機能別ソフトウェア部品の再利用を支援する組込みソフトウェア開発方式を提案し，プロタイピングにより検証した．本方式では，状態遷移表の編集と部分的なプログラミングを行なうことによって，機能別ソフトウェア部品を再利用しながら必要とする組込みソフトウェアが得られる．

今後は，実践的な組込みシステム開発事例においても本方式が有効であるかの検証を行ない，コンパイル方法の最適化についても検討したい．

参　考　文　献

1) Gamma, E., Helm, R., Johnson, R., Vlissides, J.: Design Patterns: Elements of Reusable Object-Oriented Software, Addison-Wesley (1995).
2) Fowler, M.: Analysis Patterns: Reusable Object Models, Addison-Wesley (1997).
3) 青山幹雄, 中所武司, 向山　博:コンポーネントウェア, 共立出版(1998).
4) 岡本鉄兵, 清尾克彦, 小泉寿男:制御用組み込みソフトウェアのトップダウン型設計方式, 情報処理学会第 59 回全国大会講演論文集(第 1 分冊), pp.279-280 (1999).
5) 岡本鉄兵, 小泉寿男:XML を用いた組込みソフトウェアの動的仕様記述言語 EML の提案:情報処理学会第 61 回全国大会講演論文集(第 3 分冊), pp.9-10 (2000).
6) 渡辺政彦:拡張階層化状態遷移表設計手法 Ver.2.0, 東銀座出版社 (1998).
7) Douglass, B.:Real-Time UML: Developing Efficient Objects for Embedded Systems, Addison-Wesley (1997).
8) 桝本　清:組み込みソフトウェア向け状態遷移表コンパイラの作成, Interface 2001 年 5 月号, CQ 出版社, pp.166-174(2001).